CONSTRAINT OPTIMAL SELECTION TECHNIQUES (COSTs) FOR A CLASS OF LINEAR PROGRAMMING PROBLEMS

by

TAI-KUAN SUNG

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

AUGUST 2006

To my parents, sisters, brother, and wife.

ACKNOWLEDGEMENTS

The dissertation work would not have been possible without the support, guidance, and advice from my supervising professors Dr. H.W. Corley and Dr. Jay M. Rosenberger. I deeply appreciate their encouragement, help, and understanding.

I also would like to thank the remaining members of the Committee, Dr. R. C. Baker, Dr. B. Boardman, and Dr. D. H. Liles for their valuable comments on my research and for taking the time to be on the Committee.

Many thanks go to Dr. V. C. Chen, Dr. J. K. Rogers, Dr. I. Dragan, Dr. Chien-Pai Han, and Dr. Seoung Bum Kim. Their courses have benefited me greatly during my doctoral studies. Also, thanks to all my friends in the COSMOS lab Huiyuan Fan, Heesu Hwang (Peter), Patty Insuwan, Prashant Kumar, Dr.Ventaka Pilla, Prattana Punnakitikashem (Sandy), Dachuan Shih (Thomas), Sheela Siddappa, Panita Suebvisai, Duraikannan Sundaramoorthi, Siriwat Visoldilokpun (Pop), and Dr. Aihong Wen, as well as Jawahar Veera in the Robotics lab. They made my life in COSMOS fulfilling both academically and socially.

Finally, I would like to express my gratitude to my parents, my brother Bruce (Po-Kuan), my elder sisters Helen, Susan, and Shu-Ling, and my wife Shu-Chen (Emily). I am especially indebted to my late father for his support and sacrifice. I deeply wish that I had the chance to hug him again, to thank him for everything he did for me, and to tell him how much I miss him. Special thanks are reserved for my dear

mother for building a family that keeps all her children bound strongly together, even though they all now have their own families. Without the boundless support from her, my sisters, and my brother, I would have not dreamed of pursuing a Ph.D. Saving the best for last, I can never express my full gratitude to my loving wife, who is expecting our first child. Her company and care has made my time in Texas the most joyful of my life.

July 7, 2006

ABSTRACT

CONSTRAINT OPTIMAL SELECTION TECHNIQUES (COSTs)

FOR A CLASS OF LINEAR PROGRAMMING PROBLEMS

Publication No. _____

Tai-Kuan Sung, PhD.

The University of Texas at Arlington, 2006

Co-Supervising Professor: H.W. Corley

Co-Supervising Professor: J. M. Rosenberger

This dissertation describes two classes of Constraint Optimal Selection Techniques (COSTs). An algorithm of each type is developed for solving nonnegative linear programming problems. In addition, geometric interpretations of these new algorithms are given, computational results for some large-scale problems are provided, and directions for future research are discussed.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Description of the Problem

Linear programming (LP) represents a mathematical model for solving numerous practical industrial problems such as the optimal allocation of resources. The standard linear programming model is formulated as

$$\textbf{(P)} \qquad \text{maximize } z = c_1x_1 + \ldots + c_nx_n \tag{1.1}$$

subject to

$$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i, \quad i = 1, \ldots, m \tag{1.2}$$

$$x_j \geq 0, \quad j = 1, \ldots, n, \tag{1.3}$$

where the variable x_j, $j = 1, \ldots, n$ may be interpreted as the amount of product j (or activity j) requiring a_{ij} units of resource i to be produced (or allocated) when there are only b_i, $i = 1, \ldots, m$ units of resource i available. The objective is to maximize the gain (or profit) $c_1x_1 + \ldots + c_nx_n$, where c_j, $j = 1, \ldots, n$ represents the profit per unit of product j. In any given problem (**P**), the c_j, a_{ij}, and b_i are known constants and the variables x_j must be determined to maximize the total gain z. In many practical problems, the known constants b_i and c_j are positive numbers, while the given constants a_{ij} are nonnegative numbers, in which the problem (**P**) is called a nonnegative linear programming (NNLP) problem.

LP has been studied for over fifty years, and the applications are pervasive throughout industry. Indeed, a major proportion of all scientific computation is devoted to linear programming. The simplex algorithm, developed in the 1947 by Dantzig [12], provided the first effective solution procedure and remains the underlying algorithm utilized by most commercial linear programming packages. Unfortunately its computational complexity is not polynomial in the worst case, even though it exhibits polynomial complexity on average. Khachian's ellipsoid algorithm [23] solves linear programming problems in polynomial time, but it performs poorly in practice. More recently, Dikin [16], [17], Karmarkar [22], and others, developed a polynomial projection approach that evolved into various polynomial interior-point barrier-function methods [13], [40], [41] used in some large-scale applications. However, in contrast to pivoting algorithms, such interior-point methods do not allow efficient post optimality analysis for solving the binary and integer models common in industrial applications.

Unfortunately, the simplex and other LP algorithms are not always adequate for many applications. In particular, emerging technologies require computer solutions in real time and algorithms with efficient memory usage for problems involving millions of constraints or variables. Examples include instantly updating airline schedules as weather conditions and passenger loads change [36], finding an optimal driving route using real-time traffic data from global positioning satellites [15], and detecting problematic repeats in DNA sequences [27]. Many such applications of linear programming are nonnegative linear programs (NNLPs), so only NNLPs are considered in this dissertation. Efficient new algorithms are developed for their solutions.

1.2 Objectives of the Work

According to Bixby [7] and Todd [40] recently, as well as Koopmans [25] earlier, there is no single best algorithm for LP. For any existing approach developed so far, one can always devise a problem for which the approach performs very poorly. We attempt to rectify this situation with this research. The major objective of the work is to develop new algorithms that outperform current algorithms, especially in solving an NNLP.

Each new algorithm here for solving an NNLP is termed a Constraint Optimal Selection Technique (COST). Indeed, COST represents a unifying framework for a new class of algorithms. Two classes of COSTs, Prior and Posterior, are defined. Prior COSTs use global information of relaxed LP problems with only a subset of constraints (1.2) of (**P**). Posterior COSTs utilize local information at current optimal solutions of relaxed problems in addition, as well as information about the above subset of constraints. Two new algorithms, one for each class of COST, are implemented and tested. Both of these algorithms outperform existing methods in the large-scale computational experiments performed in this research.

1.3 Related Work

Numerous approaches for solving LP have been developed since the simplex method was first introduced. An overview of approaches for solving LP is given in Todd [40]. An overview of computational aspects in solving LP is given in Bixby [7]. Corley et al. [11] proposed a COST using a criterion named the cosine criterion to solve (**P**). This algorithm is a prior COST and sequentially adds a constraint in (1.2) most

parallel to the objective function (1.1) to a relaxed version of (**P**) not including the constraint.

Recent work in Vieira et al. [43], Trigos et al. [42], and Stojkovic et al. [39] have also used the cosine criterion for (**P**) when the nonnegativity restrictions (1.3) are included in constraint set (1.2) to get an initial basis for the simplex algorithm. As a consequence, they reduce the number of required simplex iterations, each of which involves all constraints. However, these approaches must initially eliminate any redundant constraints.

1.4 Overview of the Dissertation

Chapter 2 is a literature survey. It begins with the simplex method, then the ellipsoid method and interior-point methods. Large Scale LP problems with special structure and methods to optimize them are also discussed. The types of testing problems used in this research are next described. Finally, the cosine cutting plane algorithm of (4.1) is presented since it is generalized here to COSTs and represents the prototype of these methods.

Chapter 3 describes COSTs and gives details about the specific prior and posterior algorithms developed here. Chapter 4 summarizes results of the computational experiments, and finally conclusions are presented Chapter 5.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

The major aspects of linear programming relevant to this research are summarized in this chapter. All are not used here, but all have influenced this work, which may improve them. In Section 2, the necessary definitions and notation are introduced. In Section 3, the standard Simplex Algorithm is presented, while the Ellipsoid and Interior-Point Methods are discussed. Then Section 5 covers some topics in Large-Scale Linear Programming since these are the types of problems for which the results here were developed. In Sections 6, 7, 8 cutting-plane methods, active-set methods, and integer programming are described, respectively. Finally Section 9 focuses on the cosine approach to selecting LP constraints since it was the motivation for this work.

2.2 Preliminary Definitions and Notation

2.2.1 Notation

The notation described below will be followed in general with minor deviations where appropriate. Standard definitions and notation are used, much as in Bazaraa [2] and Dantzig[13]

- Bold uppercase letters will be used to represent matrices.
- Bold lowercase letters will be used to represent vectors.

  All vectors will be column vectors unless otherwise noted, with x_i being the i^{th} component of **x**, for example.
- Lowercase letters represent scalars, including Greek letters.
- The following specific symbols are used.

R^n — n-dimensional space of real numbers.

R^n_+ — n-dimensional space of nonnegative real numbers.

c — $n \times 1$ coefficient vector $[c_1, \ldots, c_n]^T$ of the objective function.

A — $m \times n$ coefficient matrix $[a_{ij}]$ of the linear program.

B — $m \times m$ basis nonsingular matrix with the basic columns of **A**.

N — $m \times (n - m)$ nonbasic columns of **A**.

x — $n \times 1$ vector $[x_1, \ldots, x_n]^T$ usually representing the vector of unknown variables in problem (**P**).

$\mathbf{I_B}$ — $\mathbf{I_B} = \{k_1, \ldots, k_m\}$, the index set of the basic variables, where k_i is the original index of the variable that belongs to the i^{th} column of the basis **B** if we number the columns in **B** consecutively starting from 1.

p_j — For $j \in \mathbf{I_B}$, $p_j \in \{1, 2, 3, \ldots, m\}$ denotes the position number of variable j in the basis. Here j is the original column index in the list of columns of **A**. Thus if $k_j \in \mathbf{I_B}$ is the variable in the position of the basis, then $p_j = i$ for $i = 1, \ldots, m$.

$\mathbf{x_B}$ — Basic solution. The vector of basic variables in the same order as in **B**.

$\mathbf{x_N}$ — Nonbasic solution.

$\mathbf{x}^*$ — An optimal basic feasible solution of the linear program.

z_B — $z_B = \mathbf{c_B B^{-1} b}$, the objective function value given the basis **B**, where $\mathbf{c_B}$ is the row vector of the objective function coefficients of the basic variables.

$\mathbf{b'}$ — $\mathbf{b'} = \mathbf{B^{-1} b}$, the transformed right hand side for the basis matrix **B**.

$\mathbf{c_N}$ — $\mathbf{c_N} = \mathbf{c} - \mathbf{c_B B^{-1} A} = [c_1, \ldots, c_n]^{\mathbf{T}}$, the reduced cost vector.

$\mathbf{y_j}$ — $\mathbf{y_j} = \mathbf{B^{-1} a_j} = [y_j^1, \ldots, y_j^m]^{\mathbf{T}}$, the j^{th} the transformed column of **A.**

I — Identity square matrix.

$\boldsymbol{e}_{\mathbf{r}}$ — The r^{th} column of **I** where an element of one is in row r and zeros elsewhere.

2.2.2 Preliminaries

Consider the problem (**P**) of Chapter 1. A single constraint of a linear programming model $a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i$ in (1.2) can be written as $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} \leq b_i$, where $\mathbf{a}_i^{\mathbf{T}} = [a_{i1}, a_{i2}, \ldots, a_{in}]$ and $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathbf{T}}$. The set of points **x** in R^n that satisfy this constraint is called a **closed half-space**. A point that satisfies all the constraints is called a **feasible solution** or a **feasible point**. The set of all feasible points constitutes the **feasible region** or the **feasible space**.

Consider the following problem

$$\text{maximize } z = 3x_1 + 5x_2$$

$$\text{subject to } x_1 + 2x_2 \leq 3$$

$$x_1 \leq 4$$

$$x_2 \leq 6$$

$$3x_1 + 2x_2 \leq 18$$

$$x_1, x_2 \geq 0$$

The **feasible region** is shaded region in Figure 2.1

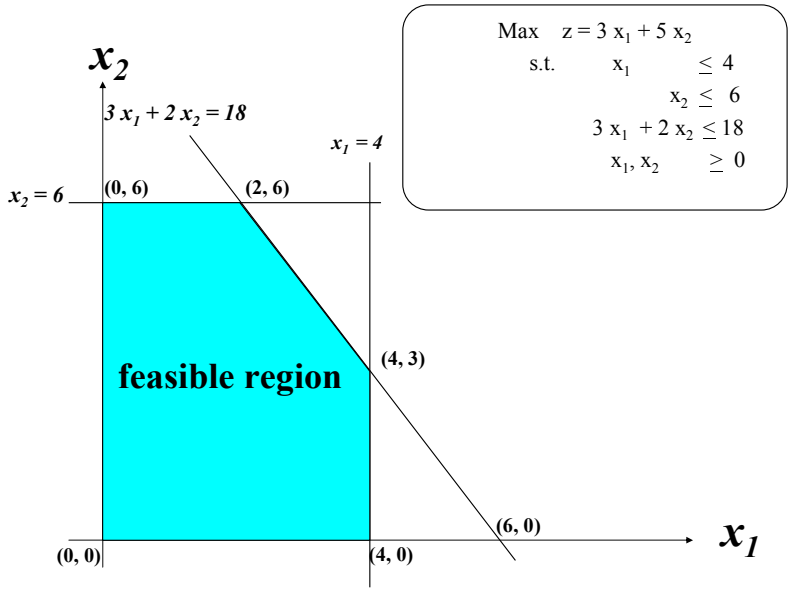

Figure 2.1 A Graphical LP Example

A **polyhedron** is the intersection of a finite set of closed half-spaces. It is a set that can be described in the form $\{\mathbf{x} \mid \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \in R^n\}$, where **A** is a $m \times n$ matrix, $\mathbf{b} \in R^m$, and $\mathbf{Ax} \geq \mathbf{b}$ means that constraints are inequality. The set $\{\mathbf{x} \mid \mathbf{A}\ \mathbf{x} = \mathbf{b}, \mathbf{x} \in R^n\}$ is call a **hyperplane**. A **polytope** is a bounded polyhedron.

A **convex combination** of the points $\mathbf{x}_1, \ldots, \mathbf{x}_k \in R^n$ is a linear combination $\lambda_1\mathbf{x}_1 + \ldots + \lambda_k\mathbf{x}_k$ such that $\sum_1^k \lambda_i = 1$, and $\lambda_i \geq 0$ for all $i = 1, \ldots, k$. A set S is a **convex set** if for any $\mathbf{x}, \mathbf{y} \in S$ and all $\lambda \in [0, 1]$, then $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \in S$. That is, any convex combination **x** and **y** is a member of S. A set $C \subseteq R^n$ is a **cone** if and only if for all $\mathbf{x} \in C$, $\lambda\mathbf{x} \in C$ for all nonnegative $\lambda \in R^n$.

Let the polyhedron $P = \{\mathbf{x} \in R^n \mid \mathbf{Ax} \geq \mathbf{b}\}$. The point **r** is a **ray** of P if and only if for any $x \in P$, the set $\{\mathbf{y} \in R^n \mid \mathbf{y} = \mathbf{x} + \lambda\mathbf{r}\} \subseteq P$. A ray **r** is an **extreme ray** if there

do not exist rays $\mathbf{r}_1$ and $\mathbf{r}_2$ such that $\mathbf{r}$ is a convex combination of $\mathbf{r}_1$ and $\mathbf{r}_2$. A point $\mathbf{x} \in P$ is an **extreme point** if and only if it can not be written as a convex combination of other points in P distinct from $\mathbf{x}$.

Consider a polyhedron P defined by linear equality and inequality constraints. Let $\mathbf{x}^*$ be an element of R^n. If the vector $\mathbf{x}^*$ satisfies $a_{i1}x_1 + \ldots + a_{in}x_n = b_i$, for some i in the constraint set, the corresponding constraint is said to be **active, tight,** or **binding** at $\mathbf{x}^*$.

The vector $\mathbf{x}$ is a **basic solution** if $\mathbf{x}$ is obtained by setting $n - m$ variables equal to zero and solving for the remaining variables [14]. If $\mathbf{x}$ is a basic solution with all nonnegative components, then it is a **basic feasible solution [**2]**.**

The following are equivalent:

a. $\mathbf{x}$ is a vertex of $\{ \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \}$;

b. $\mathbf{x}$ is an extreme point of $\{ \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \}$;

c. $\mathbf{x}$ is a basic feasible solution of $\{ \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \}$.

In Figure 2.1, the point (2, 6) is an extreme point.

An algorithm is **polynomial** if:

1). The number of arithmetic operations is bounded by a polynomial function of the number of inputs and the size of the inputs.

2). The space required is a polynomial function of the size of the inputs [28].

2.3 The Simplex Method

2.3.1 Discussion

The extreme points of (**P**) have the following properties:

1). The set of extreme points is a convex set.

2). If (**P**) has an optimal solution, then there is at least one extreme point is an optimal solution.

3). There are a finite number of extreme points in (**P**).

4). An extreme point can be represented algebraically as a basic feasible solution to the set of constraints rewritten as equations with nonnegative slack variables added to give equality.

The simplex method introduced in this section solves (**P**) as follows. The algorithm moves geometrically from one extreme point to an adjacent extreme point by algebraically changing only one basic variable per iteration until no improvement can be found.

2.3.2 Primal and Dual Linear Programming Problems

Consider the linear programming problem in standard matrix form

$$(\mathbf{P}) \qquad \text{maximize } z = \mathbf{c}^{\mathrm{T}}\mathbf{x}$$

$$\text{subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

$$\mathbf{x} \geq 0,$$

where **x** is a $n \times 1$ vector in R^n, **A** is a $m \times n$ matrix with $m < n$, **b** is a vector in R_+^m, and **c** is a vector in R^n.

Associated with problem (**P**), called the **primal problem**, is a second minimization LP problem called the **dual problem** (**D**).

(**D**) minimize $w = \mathbf{b}^T\mathbf{y}$

subject to $\mathbf{A}^T\mathbf{y} \geq \mathbf{c}$

$\mathbf{y} \geq 0,$

where **y** is a $m \times 1$ vector in R^m, $\mathbf{A}^T$ is a $n \times m$ matrix with $m < n$, **b** is a vector in R_+^m, and **c** is a vector in R^n.

The relationships between (**P**) and (**D**) were first exploited by Lemke [26]. The main results are stated as follows.

Weak Duality Theorem: If **x** is a primal feasible solution, and is **y** is dual feasible, then $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$.

Strong Duality Theorem:

1. If either the primal or the dual linear programming problem has a finite optimal solution, then so does the other. Both achieve the same optimal objective function value.
2. If either the primal or the dual linear programming problem has an unbounded objective value, then the other has no feasible solution.

The Strong Duality Theorem has several implications. First, there is no duality gap between the primal and dual linear programs; that is, no difference between their optimal objective function value z^* and w^*. Second, the Lagrange multipliers become the vector **y** of dual variables.

Complementary Slackness: Let **x** be a primal feasible solution and **y** be a dual feasible solution. Then **x** and **y** become an optimal solution pair if and only if the complementary slackness conditions

$$\text{either } s_i = (\mathbf{Ax} - \mathbf{b})_i = 0, \text{ or } y_i = 0, \text{ for all } i = 1, 2, \ldots, m$$

and

$$\text{either } r_i = (\mathbf{c} - \mathbf{A}^T\mathbf{y})_i = 0, \text{ or } x_i = 0, \text{ for all } i = 1, 2, \ldots, n$$

are satisfied. Complementary slackness is an important relationship between the primal and dual problems. It states that if a variable in one problem is positive, then the corresponding constraint in the other problem must be binding. If a constraint in one problem is not binding, then the corresponding variable in the other problem must be zero.

2.3.3 The Primal Simplex Algorithm

Consider the following linear programming problem for a minimization problem with equality constraints where nonnegative slack and surplus variables have been added as necessary to give equations for the constraints.

(**P'**) minimize $z = \mathbf{c}^T\mathbf{x}$

subject to $\mathbf{Ax} = \mathbf{b}$

$\mathbf{x} \geq 0$,

where $\mathbf{c} \in R^n$, **A** is a $m \times n$ matrix with $m < n$, and $\mathbf{b} \in R_+^m$. A submatrix **B** of **A** is called a **basis** if the rank of **B** is *m***.** Given a basis **B** we partition **A** and write

$$\mathbf{Ax} = \mathbf{Bx_B} + \mathbf{Nx_n},$$

where **N** is the "rest" of **A**, i.e., the columns of **A** not in the basis **B**. If a basis **B** is feasible, then it defines a basic feasible solution to $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq 0$ by $\mathbf{x_B} = \mathbf{B^{-1}b}$, $\mathbf{x}_n = 0$.

The classic primal simplex method of Dantzig [12, 32] is given as follows.

Primal Simplex Algorithm (*m*, *n*, **A**, **b**, **c**)

Step 0: Initialization

Find a feasible basis **B** for (**P'**), its index set $\mathbf{I_B}$,

and initialize p_k for all $k \in \mathbf{I_B}$.

if none exist **then**

stop, (**P'**) has no feasible solution.

else

compute $\mathbf{B^{-1}}$, $\mathbf{b'} \leftarrow \mathbf{B^{-1}b}$ and initialize $\mathbf{c_B}$.

end.

Step 1: Optimality Test

Compute **the reduced cost** vector $\mathbf{c_N} \leftarrow \mathbf{c} - \mathbf{c_B}\,\mathbf{B^{-1}A}$.

if $\mathbf{c_N} \geq 0$ **then**

set $\mathbf{x_B} \leftarrow \mathbf{b'}$; $\boldsymbol{x}_\mathbf{N} \leftarrow 0$,

stop, $\mathbf{x_B}$ is an optimal basic feasible solution.

end.

Step 2: Choice of Pivot Column

Choose $j \in \{k \in N : \ c_k < 0\}$.

Compute $\boldsymbol{y_j} \leftarrow \mathbf{B^{-1}}\boldsymbol{a_j}$.

if $\boldsymbol{y_j} \leq 0$, **then**

stop, (**P'**) is unbounded.

end

Step 3: Choice of Pivot Row

Compute the least ratio $\theta \leftarrow \min \left\{ \dfrac{b'_i}{y^i_j} \mid y_{ij} \geq 0,\ 1 \leq i \leq m \right\}$, and

choose a row $l \in \mathbf{I_B}$ such that the least ratio $\theta = \dfrac{b'_{p_l}}{y^{p_l}_j}$.

set $r \leftarrow p_l$.

Step 4: Pivot Step

Set $\mathbf{B} \leftarrow \mathbf{B} + (\mathbf{a}_j - \mathbf{a}_l)\,\boldsymbol{e}_\mathbf{r}^\mathbf{T}$, $\mathbf{c_B} \leftarrow \mathbf{c_B} + (c_j - c_l)\,\boldsymbol{e}_\mathbf{r}^\mathbf{T}$,

$\mathbf{I_B} \leftarrow \mathbf{I_B} - \{l\} \cup \{j\}$

$r \leftarrow p_j$.

Compute $\mathbf{B^{-1}}$, $\mathbf{b'} \leftarrow \mathbf{B^{-1}b}$ and **go to** Step 1

The primal simplex algorithm is first developed by Dantzig [12] in 1947, and its algorithmic behavior is well understood. E. M. L. Beale [3] showed that **degeneracy** (i.e., some basic variable being zeros) might cause the simplex algorithm to **cycle** infinitely. The simplex algorithm's average and worst behavior have been studied and explained by Borgwardt [8] and Klee and Minty [24], respectively. Even though in the worst case the optimal simplex method is not polynomial, it typically requires at most 2*m* to 3*m* pivots to attain optimality [29].

The primal simplex method has undergone substantial improvement since its inception. The improvements are mainly computational to make the simplex solution method efficient and robust for a wide range of problems. For example, standard simplex codes of today almost invariably use a steepest-edge rule [18] [40] for determining the entering basic variable as opposed to Dantzig's rule.

2.3.4 The Dual Simplex Algorithm

The dual simplex method was introduced in 1954 by Lemke [26]. In effect, it solves the dual problem on the primal tableau by maintaining **dual feasibility** and

complementary slackness conditions, while seeking **primal feasibility** to achieve an optimum. At each iteration, the dual simplex method moves from a basic feasible solution of the dual problem to another improved adjacent basic feasible solution until optimality is reached, or it concludes that the dual problem is unbounded and the primal is infeasible.

Consider the linear programs problem in standard form

$$(\mathbf{P}_1) \qquad \text{minimize} \quad z = \mathbf{c}^T\mathbf{x}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathbf{x} \geq 0,$$

and the rank of $(\mathbf{P}_1)$ equals m. The dual problem is

$$(\mathbf{D}_1) \qquad \text{maximize} \quad w = \mathbf{b}^T\mathbf{y}$$

$$\text{subject to } \mathbf{A}^T\mathbf{y} \leq \mathbf{c}$$

$$\mathbf{y} \text{ free.}$$

The dual simplex method [26, 32] is now described.

Dual Simplex Algorithm (m, n, **A**, **b**, **c**)

Step 0:

Find a dual feasible basis **B** for $(\mathbf{P}_1)$, its index set $\mathbf{I_B}$,

and initialize p_k for all $k \in \mathbf{I_B}$.

if none exist **then**

stop, ($\mathbf{P}_1$) is either infeasible or unbounded.

else

compute $\mathbf{c_N} \leftarrow \mathbf{c} - \mathbf{c_B}\,\mathbf{B^{-1}A}$

end.

Step 1:

Compute $\mathbf{b'} \leftarrow \mathbf{B^{-1}b}$.

if $\mathbf{b'} \geq 0$ **then**

set $\mathbf{x_B} \leftarrow \mathbf{B^{-1}b}$; $\mathbf{x_N} \leftarrow 0$,

stop, $\mathbf{x_B}$ is an optimal feasible solution to ($\mathbf{P}_1$).

else

choose $l \in \mathbf{I_B}$ such that $\mathbf{b'}_{p_l} < 0$, and set $r \leftarrow p_l$

end.

Step2:

Compute $y^r \leftarrow \boldsymbol{e}^{\mathbf{T}}{}_r\,\mathbf{B^{-1}N}$, and $J \leftarrow \{1, \ldots, n\} - \mathbf{I_B}$.

if $y^r \geq 0$, **then**

stop, ($\mathbf{P}_1$) has no feasible solution.

else

compute the least ratio $\gamma \leftarrow \min \left\{ \frac{c_k}{y_k^r} \mid y_k^r < 0,\ k \in J \right\}$, and

choose $j \in J$ such that the least ratio $\gamma = \frac{c_j}{y_{rk}}$ and $y_r < 0$.

end

Step3:

Set $\mathbf{B} \leftarrow \mathbf{B} + (a_j - a_l)\, \boldsymbol{e}^{\mathbf{T}}{}_r$,

$\mathbf{c_B} \leftarrow \mathbf{c_B} + (c_j - c_l)\, \boldsymbol{e}^{\mathbf{T}}{}_r$,

$\mathbf{I_B} \leftarrow \mathbf{I_B} - \{l\} \cup \{j\}$ and $p_j \leftarrow r$

Step 4:

$r \leftarrow p_j$.

Compute $\mathbf{B^{-1}}$, $\mathbf{c_N} \leftarrow \mathbf{c} - \mathbf{c_B}\, \mathbf{B^{-1}A}$ and **go to** Step 1.

The dual simplex method is especially useful if a dual feasible solution but not a primal feasible solution is available. This event occurs, for example, when an optimal solution to a linear programming problem is solved, but later the problem becomes infeasible when additional constraints are added. The dual simplex algorithm is used in the COSTs of the dissertation, because they are **active-set methods** that add constraints sequentially as discussed in Chapter 3.

2.4 The Ellipsoid Method and Interior-Point Methods

Dantzig's simplex method has been constantly improved and is generalized to a robust efficient method for solving linear programming problems. It remains the underlying algorithm used by most commercial linear programming packages. However, two other important proposals, very different from the simplex method, have been developed for linear programming. These new approaches are the ellipsoid method and interior-point methods.

2.4.1 Ellipsoid Method

As previously noted, the simplex method is not a polynomial algorithm in the worst case for solving linear programming. The first polynomial algorithm for linear programming is the ellipsoid method. The ellipsoid method was first developed by Yudin and Nemirovski [44] and Shor [38] for convex nonlinear programming. It received much publicity when Khachiyan [23] used it in a polynomial algorithm for linear programming.

The basic idea of the ellipsoid method is that at each iteration, an ellipsoid is given which contains all optimal solutions. By considering the center of the ellipsoid, a hyperplane is constructed so that all optimal solutions lie on one side of the hyperplane and the center either lies strictly on the other side (a deep cut) or on the hyperplane itself (a central cut). Then a new ellipsoid is found that contains all points

in the old ellipsoid and on the correct side of the hyperplane. Though the ellipsoid method was a major theoretical advance in showing a linear programming problem is polynomial, it performs poorly in practice. It is thus never used to solve applied problems.

2.4.2 Interior-Point Methods

The simplex method moves along an edge of a polyhedron from one extreme point to an adjacent one changing only one basic variable at each iteration. There are a finite number of extreme points, so the simplex method is a combinatorial algorithm. The philosophy of interior-point methods is different. Interior-point methods move in the feasible direction which gives maximum improvement per unit distance in the objective function value after projection into a linear subspace. The feasible direction will generally be through the interior of the polyhedron.

In 1984 Narendra Karmarkar developed an interior-point method with polynomial time complexity. His algorithm uses certain projective transformations. His implementation used an affine-scaling method first introduced in 1967 by Dikin [16] and further analyzed in 1976 by Dikin [17]. After Karmarkar's work, numerous other interior-point methods have been created. Today barrier methods, not Karmarkar's projective transformations, are used in most implementations of interior-point methods [40].

If the optimal solution is unique, the interior-point methods will terminate with the unique extreme point optimal solution. However, if the optimal solution is not unique, the interior-point methods will typically terminate at a point somewhere on the optimal face other than at an extreme point. In some applications an extreme point is preferred. For example, when attempting to obtain integer solutions by using relaxed linear programming problems, it is desirable to have a basic feasible solution because it will have fewer nonzeros.

The subsequent variants of Karmarkar's method often outperform the simplex method in very large, sparse problems [35]. However, the simplex method allows efficient post-optimality analysis and thus readily adapts to branch-and-bound algorithms for the binary and integer models common in industrial applications. In contrast, interior-point methods are not suitable for such uses.

2.5 Large-Scale Linear Programming

2.5.1 Discussion

Linear programming has been increasingly applied to industrial and scientific fields. Many such applied linear programming problems have special structure. They are sparse, with the nonzero items exhibiting block angular structure. Some techniques utilizing the decomposition principle are often applied to convert these large problems into one or more problems of manageable size to solve them more efficiently. Even in a manageable size a linear programming problem with some of its

constraints possessing a special structure, the decomposition principle can also be applied to solve the problem more efficiently.

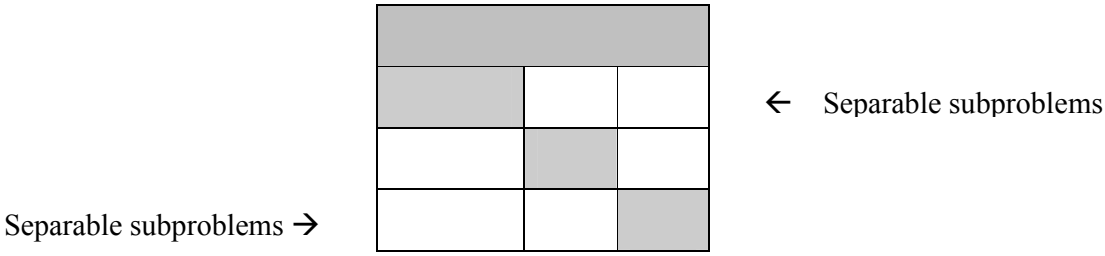

Figure 2.2 Block Angular Structure

The decomposition principle is a systematic procedure for solving large scale linear programs with special structured constraints. The decomposition strategy first partitions problems into two problems based on the constraints, one problem with general constraints named the master problem and the other with special structured constraint named the subproblem. The strategy then operates on both the master problem and the subproblem. The information is passed back and forth between these two new problems until an optimal solution to the original problem is reached.

Dantzig-Wolfe and Benders decomposition are two of these techniques. Martin [28] uses two unifying ideas, **projection** and **inverse projection**, to describe these techniques. Through projection, a system of linear inequalities is solved by replacing some of the variables with additional inequalities. Inverse projection is the dual process of projection, and it involves replacing some of the inequalities with additional variables.

The main idea of decomposition techniques for solving large-scale problems can be summarized as:

1. In a problem with an excessive number of columns, a column is generated only if its reduced cost is negative and then enters the basis.
2. In a problem with an excessive number of constraints, a constraint is generated only if it is violated by the current solution. The constraint is then added. COSTs are techniques for identifying the constraints added.

Several large scale decomposition algorithms are briefly described as follows.

2.5.2 Delayed Column Generation

Delayed column generation utilizes the idea that generating a column (variable) of the matrix **A** only after it has been determined that it can profitably enter the basis. The dual of delayed column generation's idea leads to cutting plane, or delayed constraint generation methods.

In an LP maximization problem with an excessive number of columns, a column is generated only if its reduced cost is positive, and delayed column generation then selects the column to enter the basis. It requires an efficient subroutine for identifying a variable with positive reduced cost. Delayed column generation is an inverse projection approach. It is used in Dantzig-Wolfe decomposition.

2.5.3 Dantzig-Wolfe Decomposition

Dantzig-Wolfe decomposition is a method to decompose the linear program into two sets of constraints. Delayed column generation plays as the centerpiece of Dantzig-Wolfe decomposition. Dantzig-Wolfe decomposition is very successful in applications when their subproblems have special structures and are easily optimized.

The problem

$$\begin{aligned} &\text{maximize } z = \mathbf{c}^{\mathbf{T}}\mathbf{x} \\ &\text{subject to} \quad \mathbf{A'x} \le \mathbf{b'} \\ &\qquad\qquad \mathbf{A''x} \le \mathbf{b''} \end{aligned}$$

can be rewritten as

$$\begin{aligned} &\text{maximize } z = \mathbf{c}^{\mathbf{T}}\mathbf{x} \\ &\text{subject to} \quad \mathbf{A'x} \le \mathbf{b'} \\ &\qquad\qquad \mathbf{x} \in \mathbf{P} = \{\mathbf{x} |\, \mathbf{A''x} \le \mathbf{b''}\}. \end{aligned}$$

If we let

$$\mathbf{x} = \sum_{k \in K} \lambda_k x^k + \sum_{j \in J} \mu_j r^j,\ \sum_{k \in K} \lambda_k = 1,\ \lambda_k, \mu_j \ge 0,$$

where J is the set of extreme points, and K is the set of extreme rays of the problem.

The problem becomes

$$\begin{aligned} &\text{maximize} \quad \sum_{k \in K} \lambda_k c x^k + \sum_{j \in J} \mu_j c r^j \\ &\text{subject to} \quad \sum_{k \in K} \lambda_k \, \mathbf{A'} x^k + \sum_{j \in J} \mu_j' \ \mathbf{A'} r^j \ \le \mathbf{b} \\ &\qquad\qquad \sum_{k \in K} \lambda_k = 1 \\ &\qquad\qquad \lambda_k, \mu_j \ge 0. \end{aligned}$$

Dantzig-Wolfe decomposition

1. Consider a subset of extreme points $\overline{K} \subset$ K, and extreme rays $\overline{J} \subset$ J.
2. Solve master problem ($\overline{K}$, $\overline{J}$)

maximize $\sum_{k \in K} \lambda_k c x^k + \sum_{j \in J} \mu_j c r^j$

subject to $\sum_{k \in K} \lambda_k \mathbf{A'} x^k + \sum_{j \in J} \mu_j' \mathbf{A'} r^j \leq \mathbf{b}$

$\sum_{k \in K} \lambda_k = 1$

$\lambda_k, \mu_j \geq 0$.

3. Solve subproblem: maximize $\{\mathbf{x} | \mathbf{A''x} \leq \mathbf{b''}\}$ to find additional extreme points and rays:
4. If no more extreme points and rays are found, we are done; otherwise add new points and rays to $\overline{K}$ and $\overline{J}$, and go to 2.

2.5.4 Benders Decomposition

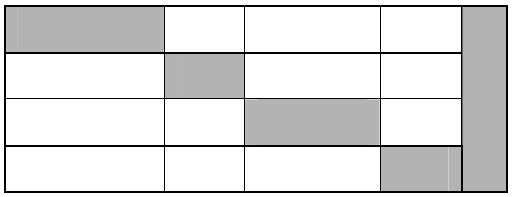

Figure 2.3 Block Angular Structure for Benders Decomposition

Benders decomposition is equivalent to Dantzig-Wolfe decomposition applied to the dual problem. Benders decomposition reduces the number of the variables at

the expense of usually adding many new constraints. The new constraints are generated only when needed, so this approach is also called delayed row generation. The key feature of Benders decomposition is that the subproblems must be solved efficiently. Benders decomposition is also used in techniques for solving integer programming [5].

Suppose the problem has the formulation:

$$\begin{aligned} \text{minimize} \quad & z = \mathbf{c}^T\mathbf{x} + \mathbf{f}^T\mathbf{y} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{D}\mathbf{y} \geq \mathbf{b} \\ & \mathbf{y} \in \mathbf{Y} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where the decision variables have been partitioned into two sets of variables $\mathbf{x} \in R^{n1}$ and $\mathbf{y} \in R^{n2}$. In particular, assume the **A** matrix has a special structure so that the problem finding an **x** is relatively easy, given a vector **y**. For example, if the **y** variables are fixed at $\mathbf{y} = \mathbf{y}'$, the constraint set is $\mathbf{A}\mathbf{x} \geq \mathbf{b} - \mathbf{D}\mathbf{y}'$ might be the constraint set for a transportation problem. The constraint set $\mathbf{y} \in \mathbf{Y}$ might be a polyhedron, a set with discrete variables (integer programming), or a set with nonlinearities.

The original problem is equivalent to:

$$\begin{aligned} \text{Minimize} \quad & z_0 \\ \text{subject to} \quad & z_0 - \mathbf{c}^T\mathbf{x} \geq \mathbf{f}^T\mathbf{y} \\ & \mathbf{A}\mathbf{x} \geq \mathbf{b} - \mathbf{D}\mathbf{y} \\ & \mathbf{y} \in \mathbf{Y} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

In the Benders decomposition framework two different problems are solved. The Benders **master problem** is defined as:

$$
\begin{aligned}
&\text{Minimize} && z_0 \\
&\text{subject to} && z_0 \geq \mathbf{f}^{\mathbf{T}}\mathbf{y} + \boldsymbol{\mu}_i^{\mathbf{T}}(\mathbf{b} - \mathbf{D}\mathbf{y}), && i = 1, \ldots, q \\
& && 0 \geq \boldsymbol{\mu}_i^{\mathbf{T}}(\mathbf{b} - \mathbf{D}\mathbf{y}), && i = q + 1, \ldots, r \\
& && \mathbf{y} \in \mathbf{Y},
\end{aligned}
$$

where $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_q$ are extreme points of the polyhedron $\{\boldsymbol{\mu} | \mathbf{A}^{\mathbf{T}}\boldsymbol{\mu} \leq \mathbf{c}, \boldsymbol{\mu} \geq 0\}$ and $\boldsymbol{\mu}_{q+1}, \ldots, \boldsymbol{\mu}_r$ are the extreme rays.

If (z_0', y') is a feasible solution to the relaxed Benders master problem, then a constraint that violates the master problem can be derived by solving the Benders **subproblem:**

$$
\begin{aligned}
&\text{maximize} && \mathbf{f}^{\mathbf{T}}\mathbf{y}' + (\mathbf{b} - \mathbf{D}\mathbf{y})\,\boldsymbol{\mu} \\
&\text{subject to} && \mathbf{A}^{\mathbf{T}}\,\boldsymbol{\mu} \leq \mathbf{c} \\
& && \boldsymbol{\mu} \geq \mathbf{0}.
\end{aligned}
$$

Benders decomposition uses delayed constraint generation and has practical applications in stochastic programming [34] and in integer-programming problems.

Benders decomposition

---

1. Find an initial $\mathbf{y} \in \mathbf{Y}$.
   Set the upper bound UB $\leftarrow \infty$, and lower bound LB $\leftarrow -\infty$.
2. **while** (UB – LB) > ε
   Solve the subproblem:

$$
\begin{aligned}
&\text{maximize} && \mathbf{f}^{\mathbf{T}}\mathbf{y}' + (\mathbf{b} - \mathbf{D}\mathbf{y})\,\boldsymbol{\mu} \\
&\text{subject to} && \mathbf{A}^{\mathbf{T}}\,\boldsymbol{\mu} \leq \mathbf{c} \\
& && \boldsymbol{\mu} \geq \mathbf{0}.
\end{aligned}
$$

   if the subproblem is unbounded
   Get a ray **u'**
   Add a cut/constraint $(\mathbf{b} - \mathbf{D}\mathbf{y})\,\boldsymbol{\mu}' \leq 0$ to the master problem
   else
   Get an extreme point **u'**
   Add a cut/constraint $z_0 \geq \mathbf{f}^{\mathbf{T}}\mathbf{y} + (\mathbf{b} - \mathbf{D}\mathbf{y})\,\mathbf{u}'$ to the master problem
   UB $\leftarrow$ mim { UB, $\mathbf{f}^{\mathbf{T}}\mathbf{y} + (\mathbf{b} - \mathbf{D}\mathbf{y})\,\mathbf{u}'$ }

end

Solve the mater problem:

minimize z_0

subject to $z_0 \geq \mathbf{f}^T\mathbf{y} + \boldsymbol{\mu}_i^T(\mathbf{b} - \mathbf{D}\mathbf{y})$, $i = 1, \ldots, q$

$0 \geq \boldsymbol{\mu}_i^T(\mathbf{b} - \mathbf{D}\mathbf{y})$, $i = q + 1, \ldots, r$

$\mathbf{y} \in \mathbf{Y}$.

if the mater problem is infeasible

Stop, the problem is infeasible**.**

end

$LB \leftarrow z_0$

end

3. **if** $(UB - LB) < \varepsilon$ **, stop**.

2.6 Cutting-Plane Methods

Cutting plane methods are applied to solve convex optimization problem of the form:

maximize $\mathbf{c}^T\mathbf{x}$

subject to $\mathbf{x} \in S$,

where $S \in \{\mathbf{x} \in R^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ is a closed convex set.

They add hyperplanes (constraints) that separate the current point from optimal points and eliminate a half-space from the feasible region. Cutting plane algorithms are sometimes used to solve integer programs.

Cutting-Plane Algorithm

Given a polyhedron $\mathbf{P}_k$, such that $S \subset \mathbf{P}_k$.

Step 1: minimize $\mathbf{c}^T\mathbf{x}$ over P_k to obtain a point in P_k.

if $\mathbf{x}_k \in S$ **then**

stop, $\mathbf{x}_k$ is optimal.

end.

Step 2: Find a hyperplane $\mathbf{H}_k$ separating the point $\mathbf{x}_k$ from S,

that is, find $\mathbf{a}_k \in R^n$, $\mathbf{b}_k \in R^1$.

such that $S \subset \{\mathbf{x}|\mathbf{a}_k^T\mathbf{x} \leq \mathbf{b}_k\}$, $x_k \in \{x|\ \mathbf{a}_k^T\mathbf{x} > \mathbf{b}_k\}$.

update $\mathbf{P}_{k+1} \leftarrow \mathbf{P}_k \bigcap \{x|\ \mathbf{a}_k^T\mathbf{x} \leq \mathbf{b}_k\}$.

go to Step 1.

Branch-and-bound algorithms represent a logical approach for solving integer linear programming problems. These branch-and-bound algorithms involve cutting planes of the form $\mathbf{x}_i \geq k+1$ and $\mathbf{x}_i \leq k$ for integers encompassing a noninteger value of the integer variable $\mathbf{x}_i$ with a noninteger value between k and k+1. Theoretically, a branch-and-bound algorithm will always find an optimal solution if one exists. However, some cutting-plane algorithms may not converge to an optimal solution in some problems.

An improvement of the branch-and-bound approach is the branch-and-cut method. Adding cuts before applying enumeration in the branch-and-bound algorithm could be very effective in reducing the amount of enumeration required. A branch-and-cut algorithm combines both, and it is used in many commercial codes.

Delayed column generation methods, when viewed in terms of the dual variables, can be described as cutting plane methods, or delayed constraint generation methods. That is, applying delayed column generation methods to the primal problem coincides with applying cutting plane methods to the dual problem. In a problem with an enormous number of constraints, a constraint (cut) is added only if it is violated by the current solution. Therefore, cutting plane methods require an efficient subroutine for identifying violated constraint.

COSTs are cutting plane methods using only the original constraints of (**P**) as possible cutting planes. However, general cutting plane methods may add constraints that are not in the original constraint set.

2.7 Active-Set Methods

An active-set method solves a system of inequalities by partitioning inequality constraints into two sets: active and inactive. The inactive constraints are ignored for an iteration. Gill and Murray [19], and Santos-Palomo et al recently [37] used a non-simplex active-set method for solving for linear programs. The COSTs in this dissertation are active-set methods.

2.8 Integer Programming

2.8.1 Definition

Often the solution of an optimization problem makes sense only if some unknowns are integers. **Integer-programming** problems have the general form

(**IP**) maximize $z = \mathbf{c}^T\mathbf{x}$

subject to $\mathbf{Ax} \leq \mathbf{b}$

x_j = integer, if $j \subset \mathbf{I}$,

where **I** is a subset of {1, …, *n*}. If **I** = {1, …, *n*}, then the problem is called a **pure integer programming problem**. If **I** is a proper subset of {1, …, *n*}, then the problem is called a **mixed integer programming problem**.

2.8.2 0-1 Integer-Programming Problems

A 0-1 integer programming problem is an integer programming problem with **x** in the set of *n*-dimensional 0-1 (binary) vectors. Many combinatorial optimization problems, such as airline **crew scheduling problems**, which are examples of set partitioning problems [21], can be modeled and solved in this form.

Let *m* be the number of flights to cover and *n* the number of tours generated. The pure IP formulation of the crew scheduling problems is:

$$\text{minimize } z = \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j = 1, \qquad \text{for all } i = \mathbf{1, \ldots}, m$$

$$x_j \in \{0, \ 1\}, \qquad \text{for all } j = \mathbf{1, \ldots}, n.$$

The x_j are 0-1 decision variables that indicate if the tour j is used. The coefficient a_{ij} equals 1 if tour j includes flight i, and 0 otherwise. The predetermined constant c_j is the cost of tour j. The number of coverings can be enormous even for medium-sized problems, and efficient algorithms are required for these problems.

2.8.3 Set-Covering Problems and Set-Partitioning Problems

The set-covering problem is given by:

(**SCP**) minimize **cx**

$$\mathbf{Ax} \geq 1$$

$$\mathbf{x_j} = 0 \text{ or } 1 \text{ for } j = 1, \ldots, n.$$

When the inequalities are replaced by equations the problem is called the **set-partitioning problem**, or when all of the inequalities are replaced by $\leq$ constraints, the problem is called the **set-packing problem.**

Many real world problems are modeled and solved by set-covering, set-partitioning, or set-packing problems. For example, disruptions in airline transportation systems can prevent airlines from executing their schedules as planned. Rosenberger et al. [36] present an optimization model for aircraft recovery (ARO)

that reschedules flight legs and reroutes aircraft by minimizing an objective function involving both rerouting and cancellation costs.

The **aircraft recovery problem** is modeled as a set packing problem in which each leg is either in a route or cancelled. Consider a set of aircraft Φ, a set of disrupted aircraft $\Phi^* \subseteq \Phi$, and a time horizon (t_0, T). For each $\phi \in \Phi$, let $F(\phi)$ be the initial route of aircraft ϕ, and let $F = \cup_{\phi \in \Phi} F(\phi)$ be the set of all legs in any initial route, for each $f \in F$, let b_f be the cost of canceling leg f, and let K_f be 1 if leg l is cancelled, or be 0 otherwise. For each aircraft $\phi \in \Phi$, let $R(\phi, F)$ be the set of maintenance feasible routes of aircraft ϕ that can be constructed from legs in F. For each route $r \in R(\phi, F)$, let the c_r be the cost of assigning route r to aircraft ϕ, and let X_r be 1 if route r is assigned to aircraft ϕ or 0 otherwise.

Let A be the set of allocated arrival slots. For each slot $a \in A$, the number of landings at a station is restricted within a time period to capacity α_a, and let *R(a)* be the set of routes that include legs that land in arrival slot a. For each route $r \in R(\phi, F)$, let H(r,a) be the set of legs r that use slot a.

Then aircraft recovery problem is a set partitioning problems:

(**ARO**) minimize $\sum_{\phi \in \Phi} \sum_{\in R} c_r X_r + \sum_{f \in F} b_f K_f$

$$\sum_{r \in R(\Phi, F)} X_r = 1, \qquad \forall \phi \in \Phi$$

$$\sum_{f \in r} X_r + K_f = 1, \qquad \forall f \in F$$

$$\sum_{f \in r} |H(r, a)| X_r \leq \alpha_a, \qquad \forall a \in A$$

$$X_r \in \{0, 1\}, \qquad \forall r \in R(\phi, F), \phi \in \Phi$$

$$K_f \in \{0, 1\}, \qquad \forall f \in F.$$

2.9 The Cosine Approach

2.9.1 Discussion

Corley et al. [11] develop a cosine criterion and present an elementary cosine-based algorithm (to be called COS in Chapter 3) to extend the simplex method when (**P**) is assumed to have an optimal solution. Several small examples indicated the potential of COS, as well as some limitations. COS is an example of an algorithm that chooses a single violated constraint to add to a solved relaxed version of LP. This single constraint is chosen as the one most likely to be binding in an optimal solution to LP from certain considerations.

2.9.2 Related Research

Several researchers also propose methods similar to COS. Murshed et al. [30] incorrectly states that the *n* nonredundant constraints with maximum cosine criterion determine an optimal extreme point. However, a counterexample is given in Corley et al. [11].

Recent work in Vieira et al. [43], Trigos et al. [42], and Stojkovic et al. [39] have used the cosine criterion for (**P**) when the nonnegativity restrictions (1.3) are included in constraint set (1.2) to get an initial basis for the simplex algorithm. As a consequence, they reduce the number of required simplex iterations, each of which involves all constraints. However, these approaches must initially eliminate any redundant constraints. COS solves (**P**) by keeping constraints (1.2) and (1.3) separate and then applying the cosine criterion only to the constraints of set (1.2). The result is a series of relaxed problems involving only a fraction of the original constraints (1.2). Redundancy difficulties are automatically eliminated in the process.

The literature relevant to LP and COS has been given in this chapter. In the next chapter the notion of a COST will be formally defined, and efficient new COSTs will be developed.

CHAPTER 3

CONSTRAINT OPTIMAL SELECTION TECHNIQUES (COSTs)

3.1 Introduction

In this chapter Constraint Optimal Selection Techniques (COSTs) are described. The rationale for a COST is that the solution to LP is determined by relatively few constraints satisfied as equalities: n such constraints for the n variables in the constraints (1.2). The difficulty lies in determining the correct ones active at optimality. From empirical evidence, two factors appear to influence the likelihood of a constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \le b_i$ being among these n:

Factor I - the angle of its normal vector $\mathbf{a}_i$ with $\mathbf{c}$ of the objective function in a manner similar to a class of problems solved by the Schwarz inequality in abstract Hilbert spaces [31],

Factor II - its efficiency as a cutting plane for at solution $\mathbf{x}^*$ to a relaxed version of (**P**) without some of the constraints in (1.2), including $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \le b_i$ itself.

We first consider Factor I. The angle that the normal vector $\mathbf{a}_i$ of a constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \le b_i$ in (1.2) forms with the normal vector $\mathbf{c}$ of the objective function is measured by the *cosine* of the angle between $\mathbf{a}_i$ and $\mathbf{c}$. Denote this cosine as $\cos(\mathbf{a}_i, \mathbf{c}) = (\mathbf{a}_i^{\mathrm{T}}\mathbf{c}) /$

($\|\mathbf{a}_i\| \, \|\mathbf{c}\|$). Geometrical considerations, including that of Naylor and Sell [31], suggest that a constraint $\mathbf{a}_i^T\mathbf{x} \leq b_i$ with a larger cosine value is more likely to be binding at an optimal extreme point of (**P**) than one with a smaller value. At a given solution $\mathbf{x}^*$ the selection of a single violated constraint with a largest cosine is called the **cosine criterion**, as previously stated in Section *2.9.1*. In Figure 3.1 below in two dimensions, the two constraints with normal vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ determine the optimal extreme point (2, 6) since $\mathbf{a}_1$ and $\mathbf{a}_2$ most parallel to **c**.

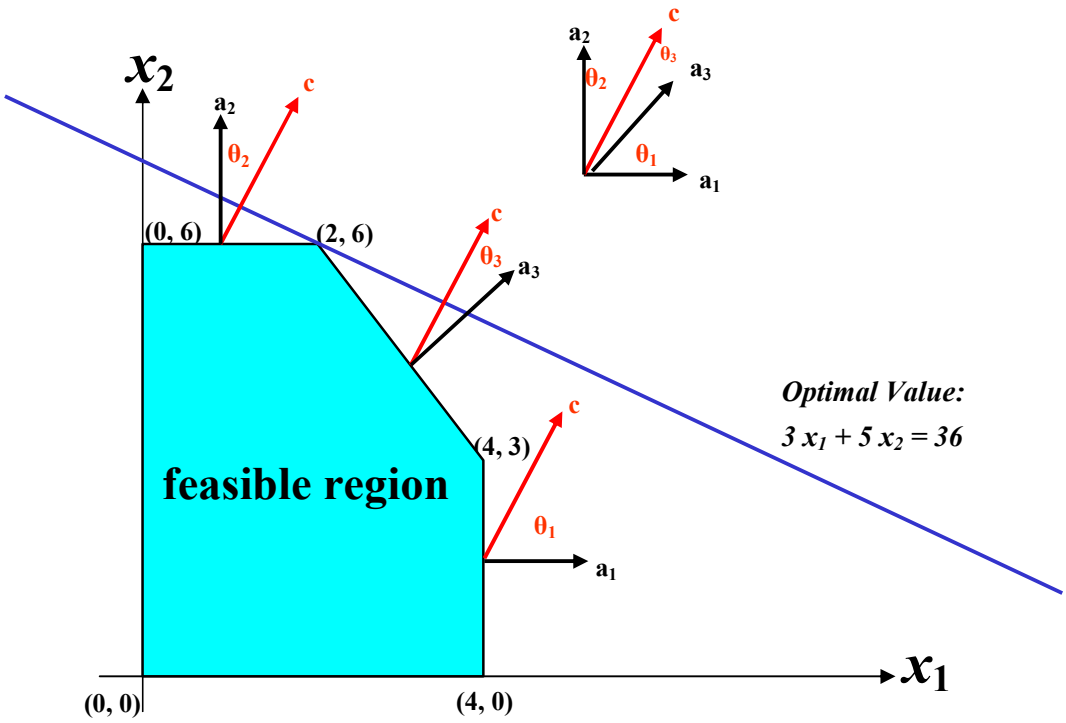

Figure 3.1 Factor I

Previous work involving the cosine criterion includes Murshed et al. [30], who incorrectly state that the n nonredundant constraints with maximum cos($\mathbf{a}_i$, **c**)

from either (1.2) or (1.3) determine an optimal extreme point. However, the three-dimensional counterexample

$$\text{maximize} \quad z = -x_1 - x_2 + 20x_3$$

subject to

$$\begin{aligned} x_1 + x_2 - 20\,x_3 &\leq 120 \\ -x_1 + x_2 + x_3 &\leq 4 \\ x_1 - x_2 + x_3 &\leq 5 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

with solution $z = 179/2$, $x_1 = 1/2$, $x_2 = 0$, $x_3 = 9/2$ is given in Corley et al. [11].

Corley et al. [11] provides the motivation for this work. The COS algorithm for (**P**) was first developed and called the Cosine Simplex Algorithm there. COS is summarized as follows.

Let $\mathbf{P}_r$ denote a relaxed version of (**P**) without some subset of constraint set (1.2), where r denotes the number of constraint from the set (1.2) in $\mathbf{P}_r$. The constraints of $\mathbf{P}_r$ are called the **operative constraints** for $\mathbf{P}_r$, while the remainder of (1.2) are called its **inoperative constraints**. Define the sequence of relaxed problems $< \mathbf{P}_r >$ such that $\mathbf{P}_{r+1}$ has a single additional operative constraint chosen from the violated inoperative constraints of $\mathbf{P}_r$ according to the cosine criterion, where the constraint of $\mathbf{P}_1$ has a constraint from (1.2) with the minimum $\cos(\mathbf{a}_i, \mathbf{c})$. In Corley et al. [11], we start with $\mathbf{P}_1$. Each problem $\mathbf{P}_r$ yields either an optimal solution $\mathbf{x_r}^*$ or a direction $\mathbf{d_r}^*$ proving $\mathbf{P}_r$ has an unbounded objective function. An inoperative $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$

said to be violated at $\mathbf{x_r}^*$ (or at the direction $\mathbf{d_r}^*$ in the unbounded case) if $\mathbf{a}_i^T\mathbf{x_r}^* - b_i > 0$ (or $\mathbf{a}_i^T\mathbf{d_r}^* > 0$ when unbounded). The relaxed problem $\mathbf{P}_{r+1}$ adds a single additional operative constraint chosen from the violated inoperative constraints of $\mathbf{P}_r$ in the $\mathbf{x_r}^*$ and $\mathbf{d_r}^*$ cases. In COS, then $\mathbf{P}_{r+1}$ is solved via the dual simplex algorithm from LP sensitivity analysis [35]. Eventually, a solution $\mathbf{x_r}^*$ or direction $\mathbf{d_r}^*$ is obtained that satisfies all inoperative constraints for the current $\mathbf{P}_r$ yielding a solution to (**P**) or establishing that (**P**) has an unbounded objective function.

COS automatically eliminates the redundancy difficulties of Stojkovic et al. [39], Trigos et al. [42], Vieira et al. [43] and has the objective of solving (**P**) using only a fraction of the original constraints. It is an active-set [19], cutting-plane algorithm that differs substantially from non-simplex, basis-deficient active-set methods such as Gill et al. [19], Pan [33], Palomo et al. [37]. Several small examples indicate the potential of COS, as well as some limitations. For small examples COS solves (**P**) more efficiently than the simplex method, handles Klee-Minty problems easily, and prevents cycling [3].

COS is the prototypical geometric-based algorithm that invokes Factor I but not Factor II to select a single violated constraint $\mathbf{a}_i^T\mathbf{x} \leq b_i$ from (1.2) to add to a relaxed version $\mathbf{P}_r$ of (**P**) not involving $\mathbf{a}_i^T\mathbf{x} \leq b_i$. This constraint is chosen from the

inoperative constraints of $\mathbf{P}_r$ violated by its solution $\mathbf{x}^*_r$ and deemed most likely to be binding at a solution $\mathbf{x}^*$ to (**P**) according to the cosine criterion.

We next consider Factor II. The prototype of a geometric-based algorithm that invokes only Factor II but not Factor I in selecting a single violated inactive constraint to $\mathbf{P}_r$ is the well-known algorithm [1] termed **VIOL** here, which is also a standard pricing method for delayed column generation [10] in terms of the dual problem to (**P**). In VIOL, a single inactive constraint $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} \leq b_i$ with the largest violation $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} - b_i > 0$ at the solution $\mathbf{x}_r^*$ of $\mathbf{P}_r$ is added to the constraints of $\mathbf{P}_r$ to give $\mathbf{P}_{r+1}$.

3.2 Prior and Posterior COSTs

We now generalize such algorithms as COS and VIOL to a class termed **Constraint Optimal Selection Techniques** (COSTs). We consider only NNLP versions of (**P**) and develop efficient new COSTs that invoke both Factors I and II. We partition COSTs into two mutually exclusive classes: **Prior** COSTs and **Posterior** COSTs.

All Prior COSTs add a constraint to $\mathbf{P}_r$ according to a selection criterion based on the Factor I or II (or both) of the inoperative constraints. In particular, Prior COSTs use a numerical measure for each constraint such as the cosine criterion that is calculated only once, while the constraints (1.2) are being initially read by the computer program. A Prior COST is termed ***Prior*** because this selection metric for

each constraint of (1.2) is independent of the solution $\mathbf{x}^*_r$ to the current $\mathbf{P}_r$ and available before $\mathbf{x}^*_r$ is obtained. Of course, for $\mathbf{P}_r$ the added constraint is chosen from those inoperative constraints violated by the solution $\mathbf{x}^*_r$ to $\mathbf{P}_r$. In some sense, then, a Prior COST uses only global information of the constraints.

COS is the prototypical Prior COST that considers only Factor I. On the other hand, a Prior COST that does not technically involve either factor is **SUB**, so termed because the violated inoperative constraint of $\mathbf{P}_r$ with the least subscript i from the set (1.2) is chosen to be added to $\mathbf{P}_r$ for $\mathbf{P}_{r+1}$.

By comparison, Posterior COSTs compute the constraint selection metric for each violated inoperative constraint of $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ at the solution $\mathbf{x}^*_r$ to the current $\mathbf{P}_r$. Thus a Posterior COST is termed ***Posterior*** because it uses local information for all violated inoperative constraints after $\mathbf{x}^*_r$. In effect, Posterior COSTs use both global and local information about the constraints. Again, Posterior COSTs can invoke both Factors I and II. **VIOL** is the prototypical Posterior COST considering only Factor I.

For problems with many constraints, a Prior COST appears to takes less CPU time to solve (**P**) on the average than a Posterior COST (such as the Prior COST RAD and the Posterior COST VRAD defined below), because the prior constraint selection metric for each constraint of set (1.2) is determined before the solutions to the sequence $\mathbf{P}_r$. In contrast, a Posterior COST calculates its constraint selection metric

for all violated inoperative constraints at $\mathbf{x}^*_r$. These computations for each $\mathbf{P}_r$ take up more CPU time on the average than would a Prior COST.

On the other hand, a Posterior COST appears to take fewer constraints on the average to solve (**P**) than a Prior COST to achieve a solution by using specific local information at $\mathbf{x}^*_r$ in a constraint selection metric computed for each violated inoperative constraint in a given $\mathbf{P}_r$. In the next section we present two efficient algorithms, one in each class, that illustrate this point.

3.3 The Prior COST RAD

The radial algorithm (Prior COST RAD) is presented in *Section 3.3.1*, and then its geometric interpretation is given in *Section 3.3.2.*

3.3.1 Statement of RAD

We now state the **Prior COST RAD**, which involves both Factors I and II. RAD's constraint selection criterion obtains $\mathbf{P}_{r+1}$ by adding an inoperative constraint of $\mathbf{P}_r$ that is violated by a solution $\mathbf{x}^*_r$ of $\mathbf{P}_r$ and that maximizes $(\mathbf{a}_i^{\mathbf{T}}\mathbf{x} / b_i)$ among all inoperative constraints of $\mathbf{P}_r$ for NNLP's. Recall that $b_i > 0$ for NNLP's.

Define $RAD(\mathbf{a}_i,\ b_i,\ \mathbf{c},\ \mathbf{x}) = \left\{ \dfrac{\mathbf{a}_i^{\mathrm{T}}\mathbf{x}}{\mathrm{b}_i} \middle| \mathbf{a}_i^{\mathrm{T}}\mathbf{x}^* > \mathrm{b}_i \right\}$. So RAD seeks $j \in \arg \max_i\ RAD(\mathbf{a}_i,\ b_i,\ \mathbf{c},\ \mathbf{x}_r^*)$ which is equivalent to seeking $j \in \arg \max_i$

$\left\{ \frac{||\mathbf{a}_i||}{b_i} \cos(\mathbf{a}_i, \mathbf{c}) \middle| \mathbf{a}_i^T \mathbf{x}_r^* > b_i \right\}$ for each $\mathbf{P}_r$ since $\mathbf{c}$ is constant for all elements of $RAD(\mathbf{a}_i, b_i, \mathbf{c}, \mathbf{x}_r^*)$. Ties are broken arbitrarily. The term $\cos(\mathbf{a}_i, \mathbf{c})$ in $RAD(\mathbf{a}_i, b_i, \mathbf{c}, \mathbf{x}_r^*)$ invokes Factor I as in COS, while $\frac{||\mathbf{a}_i||}{b_i}$ considers Factor II.

RAD starts in $\mathbf{P_1}$ with a constraint having the smallest value of $\frac{\mathbf{a}_i^T \mathbf{c}}{b_i}$ among the constraints of (1.2). Described in detail below, RAD initially allows unbounded $\mathbf{P}_r$ as in COS above with a direction $\mathbf{d}^*$ determined by CPLEX, which is used to solve $\mathbf{P}_r$. RAD continues adding constraints one at a time until either **(P)** is solved or else found unbounded if matrix **A** has a zero vector as a column. An NNLP is never infeasible since the vector **0** is feasible. Section 3.3.2 gives a geometric interpretation of RAD justifying its name from **Radial COST**. Computational results in Chapter 4 demonstrate its efficiency on the test problems.

RAD Algorithm

STOP ← False.

OPERATIVE ← ϕ.

$j \in \arg\max_i \left\{ \frac{\mathbf{a}_i^T \mathbf{c}}{b_i} \right\}$

while *STOP = False* **do**

OPERATIVE ← *OPERATIVE* $\cup$ $\{j\}$.

Using the dual simplex method solve

maximize $z = \mathbf{c}^T \mathbf{x}$ (3.1)

subject to $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} \le b_i \quad \forall\, i \in OPERATIVE$ (3.2)

$\mathbf{x} \ge 0.$ (3.3)

if (3.1) - (3.3) has an optimal solution $\mathbf{x}^*$, **then**

if $\mathbf{a}_i^{\mathbf{T}}\mathbf{x}^* \le b_i, \forall\, i \notin OPERATIVE$, **then**

STOP ← *True*. The point $\mathbf{x}^*$ is an optimal solution.

else

$j \in \arg\max_i RAD(\mathbf{a}_i, b_i, \mathbf{c}, \mathbf{x}^*)$ (3.4)

end

else

(3.1) - (3.3) is unbounded with direction $\mathbf{d}^* \ge 0$ such that $\mathbf{c}^{\mathbf{T}}\mathbf{d}^* > 0$

if $\mathbf{a}^{\mathbf{T}}\mathbf{d}^* \le 0, \ \forall\, i \notin OPERATIVE$, **then**

STOP ← *True*. The problem is unbounded.

else

$$j \in \arg\max_i \left\{ \frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}{b_i} \middle| \mathbf{a}_i^{\mathrm{T}}\mathbf{d}^* > 0 \right\} \quad (3.5)$$

end

end

end

3.3.2 Geometric Interpretation of RAD

The geometric interpretation of RAD is given in two ways. The primal interpretation is presented in 3.3.2.1, and the dual interpretation in 3.3.2.2.

3.3.2.1 Primal Interpretation of RAD

RAD can be thought of as approximating each constraint $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} \le b_i$ by a sphere with a radius the length along the vector **c** from the origin to its intersection with $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} \le b_i$. Hence the answer to the maximization problem (**P**) with one spherical

approximating constraint is obvious. Choose the one with minimum radius, and the answer is the point of intersection. This interpretation gives RAD its name. Of course, the constraint is linear, not spherical with such a radius, so the solution to the actual one-constraint problem is at an extreme point. In Figure 3.2, the constraint selection criterion is to minimize the distance v_1.

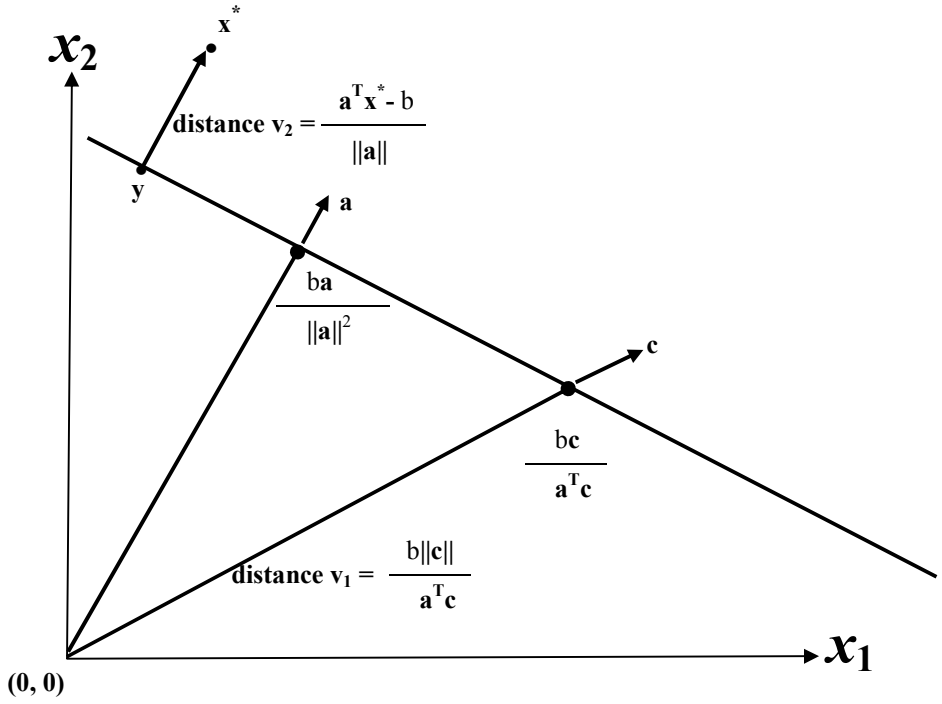

Figure 3.2 Primal Interpretation of RAD

3.3.2.2 Dual Interpretation of RAD

The geometric interpretation of RAD is also given in terms of the dual problem, which can be defined as

$$\text{maximize } w = \mathbf{b}^{\mathbf{T}}\mathbf{y} \tag{3.6}$$

$$\text{subject to} \quad \mathbf{A}^{\mathbf{T}}\mathbf{y} \geq \mathbf{c} \tag{3.7}$$

$$\mathbf{y} \geq 0. \tag{3.8}$$

where $\mathbf{y}$ is an $m \times 1$ vector of dual variables; $\mathbf{A}^{\mathbf{T}}$ is an $n \times m$ matrix $[a_{ij}]$ with $1 \times m$ row vectors $\mathbf{a}^{\mathbf{T}}_i$, $i = 1, \ldots, n$; $\mathbf{b}$ is an $m \times 1$ vector; $\mathbf{c}$ is an $n \times 1$ vector; and $\mathbf{0}$ is an $m \times 1$ vector of zeros.

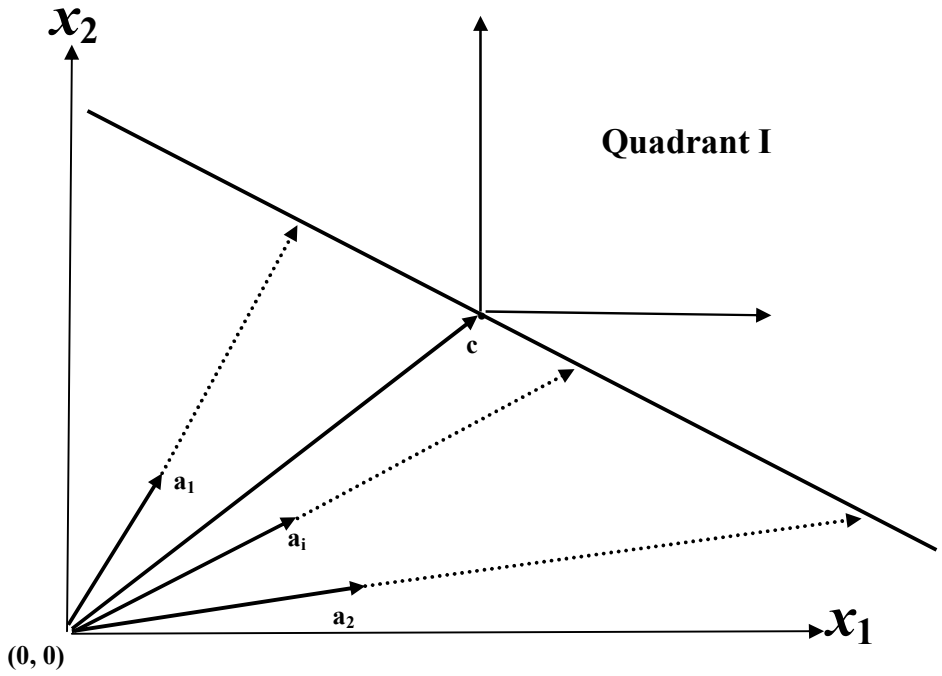

Figure 3.3 Dual Interpretation of RAD.

Consider the vectors $\mathbf{c}$ and $\mathbf{a}_i^{\mathbf{T}}$ $i = 1, \ldots, n$ in Figure 3.3. A feasible solution $\mathbf{y}$ to the dual problem is a nonnegative linear combination of the vectors $\mathbf{a}_i^{\mathbf{T}}$ that is greater than or equal to the vector $\mathbf{c}$, that is, any nonnegative combination of $\mathbf{a}_i^{\mathbf{T}}$'s to

put us in the first quadrant for which the vector **c** is the origin. The dual problem seeks to find the least cost solution in which the cost of using a vector $\mathbf{a}_i^{\mathbf{T}}$ is vectors $\mathbf{b}_i$. Now consider the following similar problem. Instead of trying to find a nonnegative combination to reach the first quadrant for which vector **c** is the origin, however, attempt to find a nonnegative combination to reach the hyperplane perpendicular to the vector **c**. The length of $\mathbf{a}_i$ projected onto **c** is $||\mathbf{a}_i||\cos(\mathbf{a}_i, \mathbf{c})$, so the number of times we need to use $\mathbf{a}_i$ to reach the hyperplane is $\frac{\|\mathbf{c}\|}{\|\mathbf{a}_i\|\cos(\mathbf{a}_i,\mathbf{c})}$. Substituting for $\cos(\mathbf{a}_i,\mathbf{c})$, the cost becomes $\frac{b_i\|\mathbf{c}\|^2}{\mathbf{c}^T\mathbf{a}_i}$. Minimizing $\frac{b_i\|\mathbf{c}\|^2}{\mathbf{c}^T\mathbf{a}_i}$ is equivalent to maximizing $\frac{\mathbf{c}^T\mathbf{a}_i}{b_i}$ (i.e. the RAD criterion), since $||\mathbf{c}||^2$ is the same for each constraint *i*.

Figure 3.4 gives us simplified geometric representations of the violation value $(\mathbf{a}_i^{\mathbf{T}}\mathbf{x}^* - b_i)$ and the right hand side value b_i of the relaxed problem (3.1) - (3.3) in the RAD Algorithm (both values are divided by $||\mathbf{a}||$ in the figure). Let $\mathbf{x}^*_r$ be the current solution of $< \mathbf{P}_r >$. The violation value = $(\mathbf{a}^{\mathbf{T}}\mathbf{x}^* - b_i) = v\cdot||\mathbf{a}||$, and the right-hand side value $b = v'\cdot||\mathbf{a}||$. The shaded area is the feasible region of $< \mathbf{P}_r >$, $\mathbf{H_{ax}}^*$ is the hyperplane of the objective function with the current solution $\mathbf{x}^*$ on it, and $\mathbf{H_a}$ is the hyperplane of the the constraint $\mathbf{a}^{\mathbf{T}}\mathbf{x}^* - b_i$

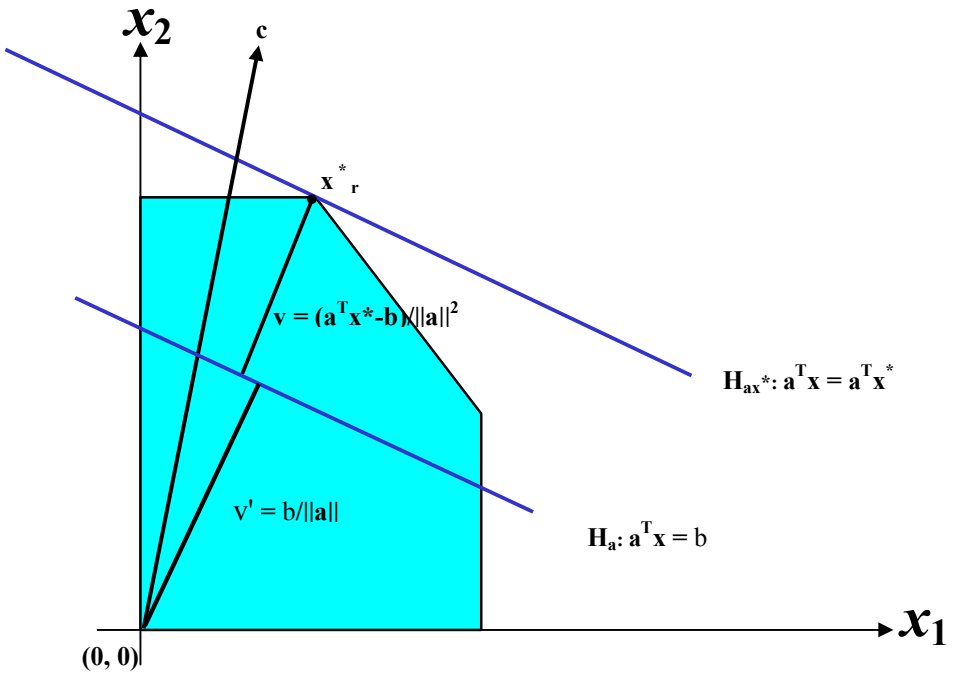

Figure 3.4 The Geometrical Representations of the Violation Value and the Right-hand-side Value of the Constraint with Normal Vector $\mathbf{a}_{r+1}$.

From the figure we can easily find how the Factor II efficiency in determined by the violated value and the right hand side value of the constraint, or cutting plane, added. Suppose the current solution is $\mathbf{x}^*$. The hyperplane $\mathbf{H}^*_c$ denotes the objective function 3.1) with $\mathbf{x}^*$ on it. The hyperplane $\mathbf{H}_a$ denotes a constraint violated by the current solution $\mathbf{x}^*$. The hyperplane $\mathbf{H}_{ax^*}$ is the hyperplane parallel with $\mathbf{H}_a$ and with $\mathbf{x}^*$ on it.

The depth of cutting plane in Factor II is proportional to $v = \frac{(\mathbf{a}^T\mathbf{x}^* - b)}{\|\mathbf{a}\|}$ and is inversely proportional to $v' = \frac{b}{||\mathbf{a}||}$. Suppose the constraints in (1.2) all have vector norms $\|\mathbf{a}\|$ near in value. Then RAD provides a deeper cut by dividing cosine criterion by *b*. Heuristically, the violated constraint with largest ratio $\frac{v}{v'} = \frac{(\mathbf{a}^T\mathbf{x}^* - b)}{b}$ is preferred in RAD and all similar COSTs. We will also use this ratio $\frac{v}{v'}$ again in *Section 3.4.2.*

3.4 The Posterior COST VRAD

The Violation Radial algorithm (VRAD) is presented in *Section 3.4.1*, and its geometric interpretation is given in *Section 3.4.2.*

3.4.1 Statement of VRAD

We next define the Posterior COST termed VRAD that again invokes both Factors I and II. Denote $VRAD(\mathbf{a}_i, b_i, \mathbf{c}, \mathbf{x}) = \left\{ \frac{\mathbf{a}_i^T\mathbf{c}}{b_i} \frac{(\mathbf{a}_i^T\mathbf{x} - b_i)}{\|\mathbf{a}_i\|} \middle| \mathbf{a}_i^T\mathbf{x}^* > b_i \right\}$.

For a solution $\mathbf{x}^*_r$ of $\mathbf{P}_r$, VRAD seeks $j \in \arg\max_i VRAD(\mathbf{a}_i, b_i, \mathbf{c}, \mathbf{x}_r^*)$ which is equivalent to seeking $j \in \arg\max_i \left\{ \frac{(\mathbf{a}_i^T\mathbf{x} - b_i)}{b_i} \cos(\mathbf{a}_i, \mathbf{c}) \middle| \mathbf{a}_i^T\mathbf{x}^* > b_i \right\}$ for each $\mathbf{P}_r$ similarly to RAD. The term $\cos(\mathbf{a}_i, \mathbf{c})$ in this equivalent expression invokes Factor I and $\frac{(\mathbf{a}_i^T\mathbf{x} - b_i)}{b_i}$ expresses Factor II. Ties are broken arbitrarily.

VRAD begins with the first constraint of (1.2) in $\mathbf{P_1}$ and proceeds as in COS with the possibility of initially unbounded $\mathbf{P}_r$ with a direction $\mathbf{d}^*$-given by CPLEX. VRAD adds constraints one at a time until either **(P)** is solved or found unbounded. The name *VRAD* comes from the fact its metric for an inoperative constraint $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} \leq b_i$ of $\mathbf{P}_r$ with solution $\mathbf{x}_r^*$ involves the product of RAD's with the constraint's violation at $\mathbf{x}_r^*$ normalized by $||\mathbf{a}_i||$. Section 3.4.2 interprets VRAD, and Chapter 4 presents computational results.

VRAD Algorithm

STOP ← False.

OPERATIVE ← ϕ.

$j = 1$.

while *STOP = False* **do**

OPERATIVE ← *OPERATIVE* $\cup$ $\{ j \}$.

Using the dual simplex method solve

maximize $z = \mathbf{c}^{\mathbf{T}}\mathbf{x}$ (3.9)

subject to $\mathbf{a}_i^{\mathbf{T}}\mathbf{x} \leq b_i \quad \forall\, i \in$ *OPERATIVE* (3.10)

$\mathbf{x} \geq 0.$ (3.11)

if (3.9) - (3.11) has an optimal solution $\mathbf{x}^*$, **then**

if $\mathbf{a}_i^{\mathbf{T}}\mathbf{x}^* \leq b_i, \forall\, i \notin$ *OPERATIVE*, **then**

STOP ← *True*. The point $\mathbf{x}^*$ is an optimal solution.

else

$j \in \arg\max_i VRAD(\mathbf{a}_i, b_i, \mathbf{c}, \mathbf{x}^*)$ (3.12)

end

else

(3.9) - (3.11) is unbounded with direction $\mathbf{d}^* \geq 0$ such that $\mathbf{c}^T\mathbf{d}^* > 0$

if $\mathbf{a}^T\mathbf{d}^* \leq 0, \ \forall\, i \notin$ *OPERATIVE*, **then**

STOP ← *True*. The problem is unbounded.

else

$$j \in \arg\max_i \left\{ \frac{\mathbf{a}_i^T\mathbf{c}}{b_i} \frac{\mathbf{a}_i^T\mathbf{d}^*}{\|\mathbf{a}_i\|} \middle| \mathbf{a}_i^T\mathbf{d}^* > 0 \right\} \qquad (3.13)$$

end

end

end

3.4.2 Geometric Interpretation of VRAD

Figure 3.2 depicts the geometric interpretation of VRAD. In Figure 3.2 we have the two distance values v_1 and v_2 and the constraint $\mathbf{a}^T\mathbf{x} \leq b$ violated by the current optimal extreme point $\mathbf{x}^*$; that is, $\mathbf{a}^T\mathbf{x}^* > b$, where $\mathbf{a}$ is the normal. We seek the perpendicular distance from the violated constraint $\mathbf{a}^T\mathbf{x} = b$ and $\mathbf{x}^*$. In particular, we seek $\mathbf{y} = \mathbf{x}^* - \alpha\mathbf{a}$, $\alpha \geq 0$ in Figure 3.2 and the distance $\|\mathbf{x}^* - (\mathbf{x}^* - \alpha\mathbf{a})\| = \|\alpha\mathbf{a}\|$. The scalar α can be found by substituting $\mathbf{x}^* - \alpha\mathbf{a}$ into $\mathbf{a}^T\mathbf{x} = b$. Doing so yields $\alpha = \frac{(\mathbf{a}^T\mathbf{x}^* - b)}{\mathbf{a}^T\mathbf{a}} = \frac{(\mathbf{a}^T\mathbf{x}^* - b)}{\|\mathbf{a}\|^2}$. Substitution gives a perpendicular distance $v_1 = \frac{(\mathbf{a}^T\mathbf{x}^* - b)}{\|\mathbf{a}\|}$.

Next compute the difference in the objective function value at x* and its perpendicular projection **y** on $\mathbf{a}^T\mathbf{x}$ = b that is the normal distance v_1 from x*. We get

$$\mathbf{c}^T[\mathbf{x}^* - (\mathbf{x}^* - \alpha\mathbf{a})] = \mathbf{c}^T\mathbf{x}^* - \mathbf{c}^T(\mathbf{x}^* - \alpha\mathbf{a}) = \alpha\mathbf{c}^T\mathbf{a} = \frac{(\mathbf{a}^T\mathbf{x}^* - b)}{\|\mathbf{a}\|^2}\mathbf{c}^T\mathbf{a}.$$

Finally compute the perpendicular distance from the origin to $\mathbf{a}^T\mathbf{x}$ = b. In essence, this distance is the depth of the cut if (2) is added as a constraint. This distance is easily found to $\beta\mathbf{a}$ from be $\mathbf{a}^T\beta\mathbf{a}$ = b. Hence the distance is $v_2 = \frac{b}{\|\mathbf{a}\|}$. Dividing $\frac{v_1}{v_2}$ yields $\frac{(\mathbf{a}^T\mathbf{x}^* - b)}{\|\mathbf{a}\|}\frac{\mathbf{c}^T\mathbf{a}}{b}$, which is precisely the VRAD constraint selection metric. In other words, this metric gives the decrease in the value of the objective function at x* and its perpendicular projection on $\mathbf{a}^T\mathbf{x}$ = b per unit depth of $\mathbf{a}^T\mathbf{x}$ = b as a cutting plane.

In Chapter 4, large-scale LP problems are formulated, computational experiments are performed, and their results are presented.

CHAPTER 4

COMPUTATIONAL EXPERIMENTS

4.1 Problem Instances

For testing, a group of nonnegative LP problems were used. These problems come from some Italian railways set-covering problems in OR-library [4] contributed by Paolo Nobili [9]. We tested their relaxed LP dual problems.

Table 4.1 Test Problems

Problems	Number of Constraints	Number of variables	Number of non-zero item	Density	Number of operative constraints at optimality
RDrail507	63,009	507	409,349	0.0128	274
RDrail516	47,311	516	314,896	0.0129	267
RDrail582	55,515	582	401,708	0.0124	278
RDrail2536	1,081,841	2,536	10,993,311	0.0040	1,066
RDrail2586	920,683	2,586	8,008,776	0.0034	1,307
RDrail4284	1,092,610	4,284	11,279,748	0.0024	1,889
RDrail4872	968,672	4,872	9,244,093	0.0020	2,100

Because all column costs of the original primal problems are either one's or two's, we randomly generated the right hand side values of the dual problems to give the b_i's more than the values 1 and 2. The right-hand-side values are between 1 and 102. Those modified dual problems are described in Table 4.1. The number of constraints for these problems is between 47,311 and 1,092,610, and the densities of

these problems are between 0.0040 to 0.0128. We define two computational measures of comparison in solving (**P**) by our algorithms, the CPU time and the number of constraints the algorithm added in solving the problems.

4.2 Results

Seven algorithms are tested. They include the dual simplex method, primal simplex method, RAD, COS, and VRAD. Also tested are VIOL and SUB. The RAD, COS and SUB are Prior COSTs; the VRAD and VIOL are Posterior COSTs.

Algorithm comparisons were performed on a Dual 2.8-GHz Intel Xeon Workstation using the dual simplex algorithm of the ILOG CPLEX 8.0 software in a Linux operating system. The Presolve function in CPLEX is truned off in testing. Computational test results are summarized in Table 4.2 - 4.4.

Table 4.2 Computational Results: Primal Simples and Dual Simplex

Problems	Primal Simplex		Dual Simplex	
	CPU time	Numbers of iterations Used	CPU time	Numbers of iterations used
RDrail507	8.08	1,339	7.46	1,347
RDrail516	2.38	677	11.09	1,479
RDrail582	0.71	420	3.47	1,180
RDrail2536	2,188.00	12,368	1,612.81	7,778
RDrail2586	2,375.00	16,178	1,255.00	8,340
RDrail4284	3,545.00	21,438	8,503.00	29,231
RDrail4872	2,278.00	18,951	1,468.31	12,065
Average	1,485.31	10,195.86	1,837.31	8,774.29

Table 4.3 Computational Results: RAD, COS, and SUB

Problems	Prior COSTs											
	RAD				COS				SUB			
	CPU time	Number of constraints added	Number of constraints evaluated	Time per constraints evaluated	CPU time	Number of constraints added	Number of constraints evaluated	Time per constraints evaluated	CPU time	Number of constraints added	Number of constraints evaluated	Time per constraints evaluated
RDrail507	0.70	558	1,098,166	6.37426E-07	9.08	1,668	37,263,474	2.43670E-07	9.05	1,875	34,205,722	2.64576E-07
RDrail516	0.59	467	1,845,771	3.19650E-07	5.31	1,521	22,102,978	2.40239E-07	5.31	1,594	23,008,595	2.30783E-07
RDrail582	0.65	628	1,635,337	3.97472E-07	8.75	1,836	37,991,123	2.30317E-07	6.51	1,872	26,861,107	2.42358E-07
RDrail2536	35.07	3,153	34,283,124	1.02295E-06	609.54	9,014	2,430,159,374	2.50823E-07	593.07	9,690	2,194,909,421	2.70202E-07
RDrail2586	57.80	3,425	159,318,440	3.62795E-07	914.24	9,006	4,343,908,098	2.10465E-07	862.32	9,363	3,682,998,535	2.34135E-07
RDrail4284	96.31	4,574	90,557,569	1.06352E-06	1,194.63	12,521	3,924,404,089	3.04411E-07	1,586.42	13,673	5,180,718,327	3.06216E-07
RDrail4872	117.14	4,925	291,207,885	4.02256E-07	1,521.19	13,091	6,485,173,079	2.34564E-07	1,744.09	14,184	6,466,575,508	2.69708E-07
Average	44.04	2532.86	82,849,470.29	6.00868E-07	608.96	6951.00	2,468,714,602.14	2.44927E-07	686.68	7,464.43	2,515,611,030.71	2.59711E-07

Table 4.4 Computational Results: VRAD and VIOL

Problems	Posterior COSTs					
	VRAD			VIOL		
	CPU time	Number of constraints added	Time per constraints added	CPU Time	Number of constraints added	Time per constraints added
RDrail507	3.27	369	0.0089	5.39	594	0.0091
RDrail516	2.15	325	0.0066	3.56	530	0.0067
RDrail582	3.22	398	0.0081	5.08	624	0.0081
RDrail2536	313.21	1,690	0.185	470.17	2,560	0.184
RDrail2586	283.44	1,999	0.142	416.15	2,908	0.143
RDrail4284	510.82	2,622	0.194	742.46	3,840	0.193
RDrail4872	492.70	3,075	0.160	742.76	4,542	0.164
Average	229.40	1,496.86	0.153	340.80	2,228.29	0.153

The CPU time for an algorithm is shown under its designated column. These CPU times are likely to be further improved if our methods were an integrated part of CPLEX.

In our implementations of COS and RAD, we first calculate the COS criterion and the RAD criterion of each constraint during the problem input step, then we sort all constraints by the COS criterion and the RAD criterion respectively. The SUB criterion simply determines the least subscript of a violated inoperative constraint, and no extra sorting is required since all constraints are already ordered by their subscripts.

The operative constraints are all satisfied in $\mathbf{P}_r$, so in our implementations we do not check if they are violated in $\mathbf{P}_{r+1}$. However, an inoperative constraint not

violated at $\mathbf{P}_r$ might be violated later in $\mathbf{P}_{r+1}$. We always check whether an inoperative constraint is satisfied in $\mathbf{P}_{r+1}$.

4.3 Discussion

Once all constraints are sorted by the criterion, Prior COSTs can easily select the first violated constraint to add. During the selection Prior COSTs need not traverse all violated constraints. Only the first violated constraint is needed in the selection. In each iteration, it takes relatively the same amount of CPU time to select the next added constraint in Prior COSTs.

However, all techniques in Posterior COSTs need local information of the current solution to calculate criteria. Unlike Prior COSTs, they need to traverse all violated constraints to find the constraint to add. Hence, Prior COSTs take less time than Posterior COSTs because all constraints of (**P**) in the set (1.2) are sorted by the constraint selection criteria as (**P**) is read into the computer. Then at each $\mathbf{P}_r$ the first violated constraints of the sorted constraints is chosen to be added. For each Prior and Posterior COST, more CPU time is required in early $\mathbf{P}_r$'s because there are more violated constraints than later < $\mathbf{P}_r$'s>. It is likely that, in general, Posterior COST reduce the number of the violated constraints more quickly.

By comparing the test results, we conclude that the Prior COST RAD outperforms ILOG's CPLEX primal and dual simplex algorithms significantly in CPU

time. The CPLEX primal and dual simplex algorithms CPU times are on the average are at least a factor of 34.78 times those of RAD. Moreover, RAD adds only 0.42% of the constraints in DRail 4284, for example.

By comparison, the results of the Posterior COST VRAD with the CPLEX primal and dual simplex algorithms, the primal and dual simplex CPU times are on the average at least a factor of 6.68 times those of VRAD. Moreover, VRAD adds only 0.23% of the constraints in DRail 4284.

CHAPTER 5

CONCLUSIONS

The goal of this dissertation is to find algorithms for nonnegative linear programming that use only relatively few constraints in determining an optimal solution. This class of linear programming problems was chosen since virtually all applied problems involve NNLP's. To accomplish this objective, we defined the general notion of a Constraint Optimal Selection Technique (COST) and considered two types of COSTs termed Prior COSTs and Posterior COSTs.

Prior Costs have a constraint selection criterion that is invoked when the constraints of a problem (excluding the nonnegativity restrictions) are read into the computer. The constraints are sorted from best to worst according to this prior selection metric (i.e., one calculated before any relaxed problem is solved). A sequence of relaxed problems is thus solved in which a single inoperative constraint is added to a previously solved relaxed problem by choosing the best violated inoperative constraint in the sorted list of constraints. This procedure eventually yields a solution to the original problem.

On the other hand, a Posterior COST uses a constraint selection criterion that must be evaluated for all constraints of the problem (excluding the nonnegativity restrictions) at the optimal point of the previous relaxed problem. The violated inoperative constraint with the best value of this posterior selection metric (i.e., after each relaxed problem in the sequence is solved) is added to the previous problem.

A Prior COST termed RAD and Posterior COST VRAD were then developed. Geometric interpretations were given, and the algorithms were tested on a set of large-scale linear programs. Computational results were compared for RAD and VRAD versus the primal simplex, dual simplex, as well as the existing control Prior COST COS, an essentially random Prior COST SUB, and the existing Posterior control COST VIOL. RAD and VRAD significantly outperformed all other algorithms. RAD was faster than VRAD, while VRAD added fewer constraints than RAD. This result suggests that RAD would be more likely to solve NNLP's in essentially real time, while VRAD could solve larger problems with its need for less memory.

Future research for COSTS includes extending RAD and VRAD to general LP's and not just NNLP's. In addition, new Prior and Posterior constraint selection criterion will be developed. The idea of applying COSTs to the primal and dual simultaneously to add both constraints and columns is also an intriguing possibility.

Moreover, the COST approach should prove useful in integer and 0-1 integer programming - in branch-and-bound methods, for example. Finally, the method will be modified appropriately and applied to an increasingly popular use of linear programs, which is to optimize a linear objective function subject to convex-quadratic (not linear) constraints. Each of these quadratic constraints will be linearized to yield a large number of linear constraints, precisely a situation in which COST's excel. Convex-quadratic programs are used in models to limit variance, or risk, among other uses. Such applications are critically important as economies are subject to enormous volatility due to such factors as shifts in energy prices, inclement weather, and global terrorism.

REFERENCES

[1] Arsham, H., Artificial-variable free solution algorithms for LP models: greater-than-or-equal constraints relaxation method, Preliminaries, Phase 2, http://home.ubalt.edu/ntsbarsh/opre640a/PartII.htm.

[2] Bazaraa, M.S., Jarvis, J.J., Sherali, H.D., Linear Programming and Network Flows, third edition, John Wiley, New York, 2005.

[3] Beale, E.M.L., Cycling in the dual simplex algorithm, Naval Research Logistics Quarterly 2(1955), 269-276.

[4] Beasly, J. E., OR-Library, http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

[5] Bertsimas, D., Tsitsiklis, J.N., Introduction to Linear Optimization, Athena Scientific, 1997, 131-159.

[6] Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R. (2000): MIP: theory and practice – closing the gap. In: Powell, M.J.D., Scholtes, S., editors, System Modeling and Optimization: Methods, Theory, and Applications, Kluwer Academic Publishers, 19–49.

[7] Bixby, R.E., Solving real-world linear programs: a decade and more of progress, Operations Research 50(2002)3-15.

[8] Borgwardt, K.H., The average number of pivot steps required by the Simplex-Method is polynomial, Zeitschrift fűr Operations Research 26(1982), 1:157 – 177.

[9] Ceria, S., Nobili, P., Sassano, A., A Lagrangian-based heuristic for large-scale set covering problems, Mathematical Programming 81(1998), 215 – 228.

[10] Chvatal, V., Linear Programming, W.H. Freeman and Company, New York, 1983, 198-200.

[11] Corley, H.W., Rosenberger, J.M., Yeh, W.C., Sung, T.K. The cosine simplex algorithm, International Journal of Advanced Manufacturing Technology, 27(2006).

[12] Dantzig, G.B., Maximization of a linear function of variables subject to linear inequalities, in Activity Analysis of Production and Allocation, T.C. Koopmans(ed.), John Wiley, New York, 1951, 339-347.

[13] Dantzig, G.B., Thapa, M.N. Linear Programming 2: Theory and Extensions, Springer 2003, Chapter 4.

[14] Dantzig, G.B., Thapa, M.N., Linear Programming 1: Introduction, Springer, New York, 1997.

[15] Dare, P., Saleh, H., GPS network design: logistics solution using optimal and near-optimal methods, Journal of Geodesy, 74(2000), 467 – 478.

[16] Dikin, I.I., Interactive solution of problems of linear and quadratic programming (in Russia), Doklady Akademiia Nauk SSSR, 174(1967), 747-748. English Translation, Soviet Mathematics Doklady 8(1967), 674-675.

[17] Dikin, I.I., On the speed of an interactive process (in Russia), Upravlaemye Sistemy 12(1974), 54-60.

[18] Forrest, J.J., Goldfarb, D., Steepest-edge simplex algorithms for linear programming, Mathematical Programming 57(1992), 341-374.

[19] Gill P.E., Murray, W., A numerically stable form of the simplex algorithm. Linear Algebra and Its Applications 7(1973), 99–138.

[20] Haimovich, M., The simplex algorithm is very good! On the expected number of pivot steps and related properties of random linear programs, Technical Report. Columbia University, New York, 1983.

[21] Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., Progress in linear programming-based algorithms for integer programming: an exposition, INFORMS J. Computing 12 (2000), 2-23.

[22] Karmarkar, N., A new polynomial-time algorithm for linear programming, Combinatorica 4(1984), 373-379.

[23] Khachiyan, L.G., A polynomial time algorithm for linear programming, Soviet Mathematics Doklady 30(1979), 191-94.

[24] Klee, V., Minty, G.J, How good is the simplex algorithm. In Shisha, O., editor, Inequalities III, Academic, New York, 1972, 159-175.

[25] Koopmans, T.C., editor, Activity Analysis of Production and Allocation, Wiley, New York, 1951.

[26] Lemke, C., The dual method of solving the linear programming problem. Naval Research Logistics Quarterly, 1954, 36-47.

[27] Li, H.L., Fu, C.J., A linear programming approach for identifying a consensus sequence on DNA sequences, Bioinformatics 21(2005).

[28] Martin, R. K., Large Scale Linear and Integer Optimization: A Unified Approach, Kluwer Academic Publishers, 1999, 350-392, 657-675.

[29] McCall, E. H., Performance results of the simplex algorithm for a set of real-world linear programming models, Communications of the ACM 25(1982), 207-212.

[30] Murshed, M.M., Sarwar, B.M., Sattar, M.A., Kaykobad, M. A new polynomial algorithm for linear programming problem, NEC Research Institute, 1993.

[31] Naylor, A., Sell, G., Linear Operator Theory in Engineering and Science, Springer-Verlag, New York, 1982, 273-274.

[32] Padberg, M., Linear Optimization and Extensions, Springer, 1991, 93-130.

[33] Pan, P.Q., A basis-deficiency-allowing variation of the simplex method of linear programming, Computers and Mathematics with Applications 36(1998), 33-53.

[34] Punnakitikashem, P., Rosengerber, J.M., Behan, D.B., Stochastic Programming for Nurse Assignment, Technical Report, http://ieweb.uta.edu.

[35] Roos, C., T. Terlaky, J. P. Vial, Interior Point Methods for Linear Optimization, Springer, New York, 2006, xix-xx.

[36] Rosenberger, J.M., Johnson, E.L., Nemhauser, G.L., Rerouting aircraft for airline recovery, Transportation Science 37(2003), 408–421.

[37] Santos-Palomo, A., Guerrero-Garcia, P., A non-simplex active-set method for linear programs in standard form, Studies in Informatics and Control 14(2005).

[38] Shor, N.Z., Cut-off method with space extension in convex programming problems (in Russian), Kibernetika 13(1977), 94-95. English translation: Cybernetics 13(1977), 94-96.

[39] Stojkovic, N.V., Stanimirovic, P.S., Two direct methods in linear programming, European Journal of Operational Research 131(2001), 417-439

[40] Todd, M.J., The many facets of linear programming. Mathematical Programming 91(2002), 417-436.

[41] Todd, M.J., Theory and practice for interior-point methods, ORSA Journal on Computing 6(1994), 28-31.

[42] Trigos, F., Frausto-Solis, J., Rivera-Lopez, R.R., A simplex cosine method for solving the Klee-Minty cube, Advances in Simulation, System Theory and Systems Engineering 70(2002), 27-32.

[43] Vieira, H., Estellita-Lins, M.P., An improved initial basis for the simplex algorithm, Computers and Operations Research 32(2005) 1983-1993.

[44] Yudin, D.B., and Nemirovski, A.S., Informational complexity and efficient methods for the solution of convex extremal problems (in Russian). Ékonomika i Matematicheskie metody 12, 357-369. English translation: Matekon 13(1976), 3-25.

BIOGRAPHICAL INFORMATION

Tai-Kuan Sung was born in Taichung City, Taiwan. He received his B.S. degree in Psychology from National Taiwan University. He received his M.S. degree in Computer Science Engineering and his Ph.D. degree in Industrial Engineering from The University of Texas at Arlington. His working experience includes Second Lieutenant in Taiwan Marines (mandatory military service), high school teacher, computer programmer, and system analyst at two banks in Taiwan. His research interests include optimization, software engineering, statistics, and system analysis.